浅谈一类分治算法

南京师范大学附属中学 顾昱洲

从1D1D动态规划说起

- 一维状态，一维转移
- 形式：f(i)=max(w(j，i))　for　0⩽j<i

- 若w函数单次计算O(1)，则暴力复杂度$O(n^2)$

一类1D1D动态规划的优化

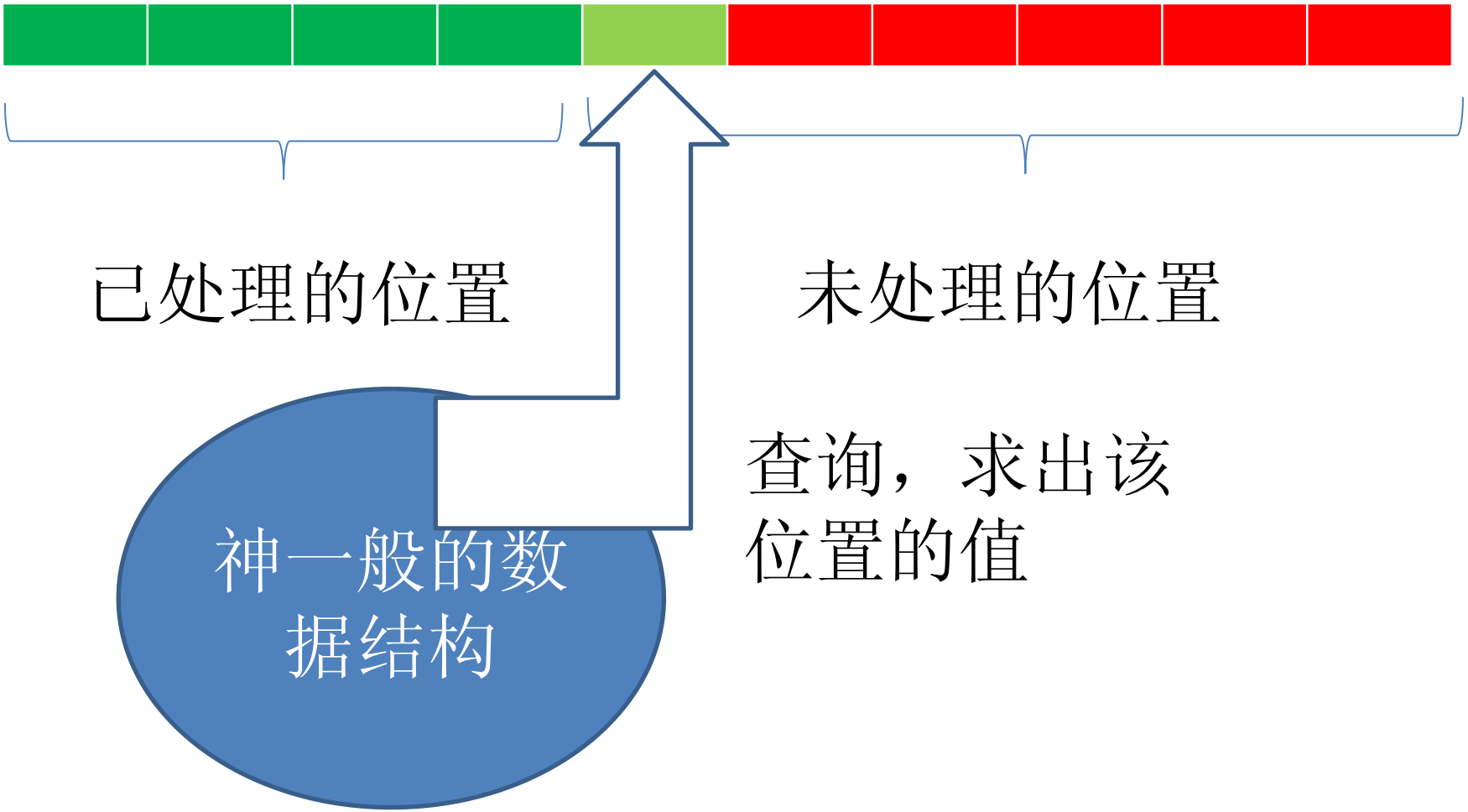
- 假设我们能够维护一个数据结构，在低于线性的时间复杂度内在线支持：
- (1) 插入 (2) 查询
- 那么可以在低于平方的时间复杂度解决原问题

Pseudo-Animation

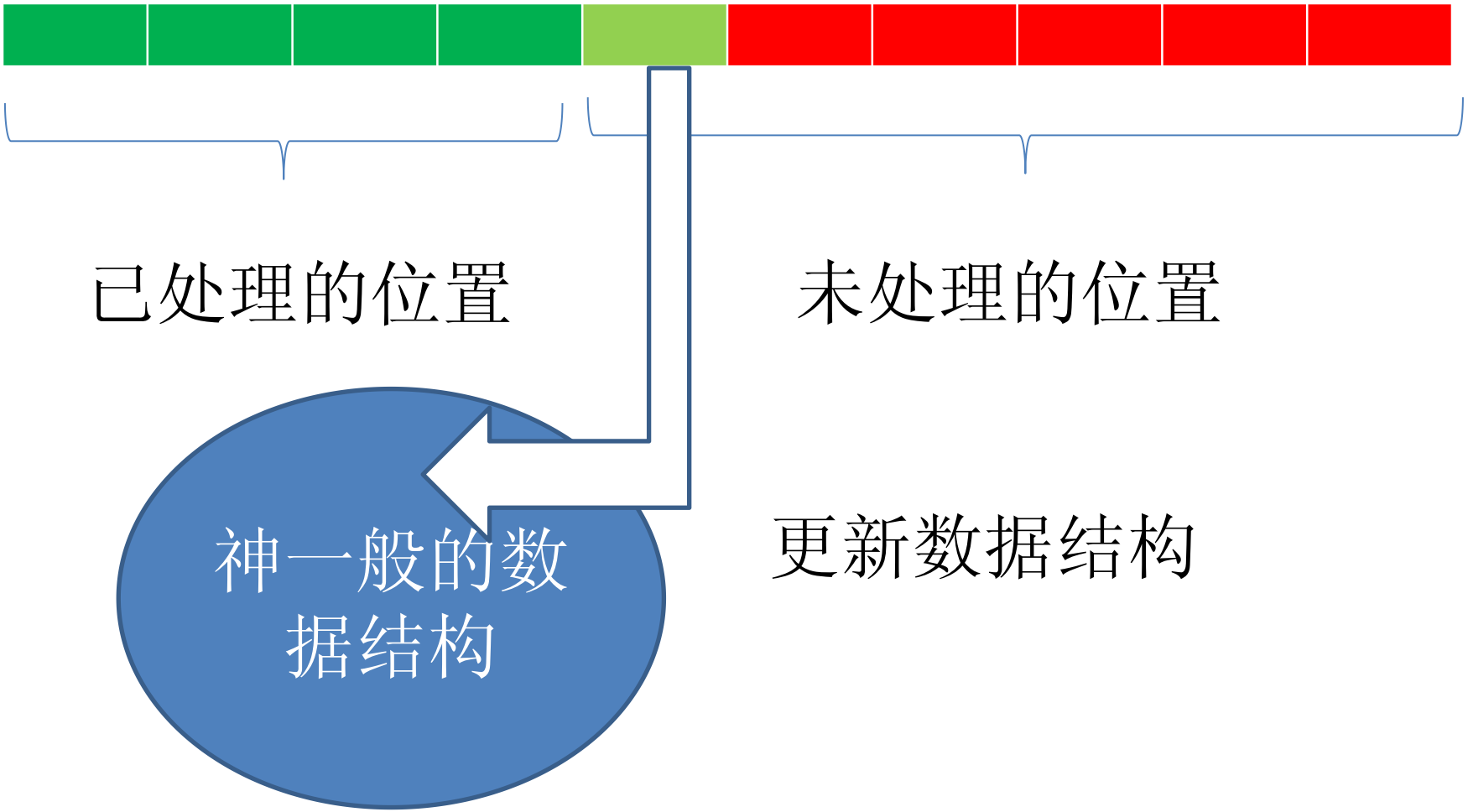
已处理的位置　　未处理的位置

神一般的数据结构

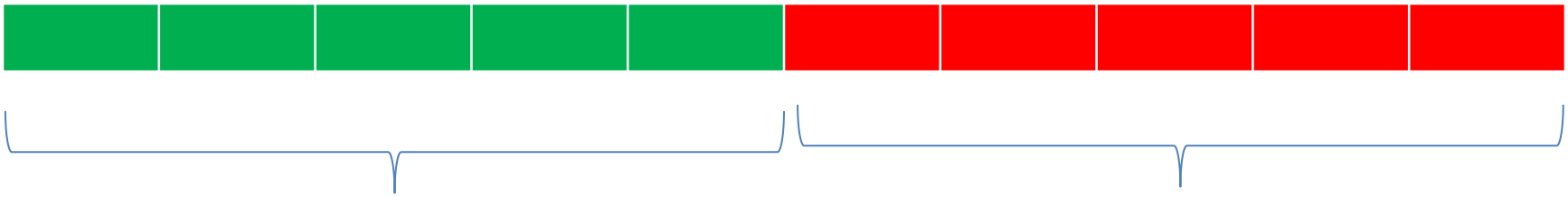
Pseudo-Animation



Pseudo-Animation



Pseudo-Animation



已处理的位置　　未处理的位置

神一般的数据结构

例子

- 最长上升子序列（树状数组）
- HNOI 2008 玩具装箱 toy（单调队列）
- CF 115 E Linear Kingdom Races（线段树）
- NOI 2007 货币兑换 cash（平衡树维护凸包）
- ACM ICPC WF 2011 F Machine Works（平衡树维护凸包）

劣势

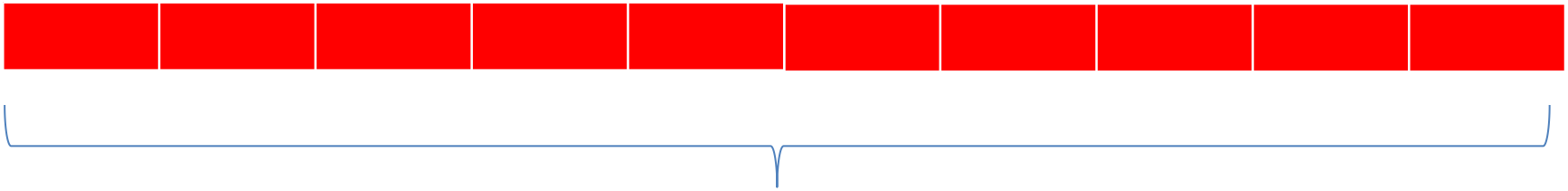
- 必须采用在线数据结构
- -> 存在性、代码量、复杂度

- 存在性：你会做吗？
- 代码量：你写得出吗？
- 复杂度：写出来你A的掉吗？

分治算法

- 当前处理区间[L，R)，中点为M
- 递归处理[L，M)
- 将[L，M)的结果建成一个数据结构，并用其更新[M，R)的结果
- 递归处理[M，R)

Actual-Animation

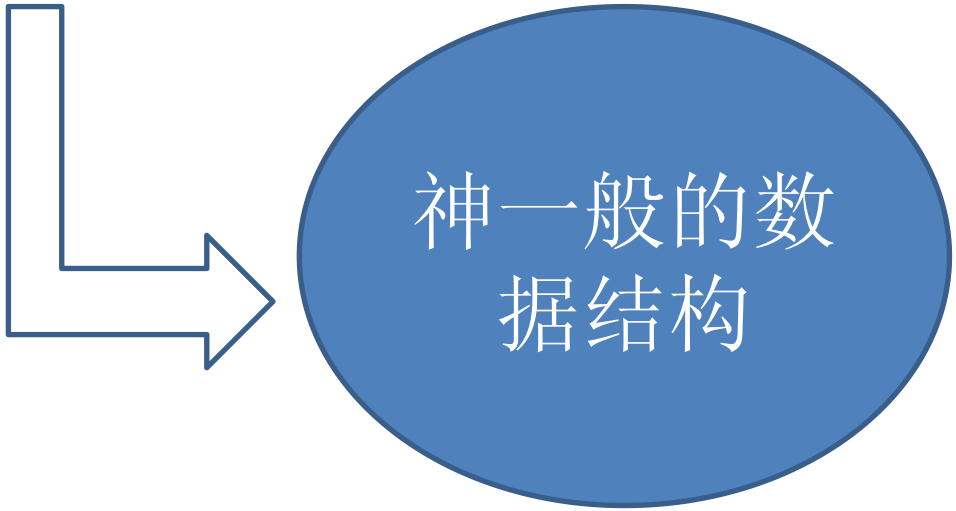


未处理的位置

递归处理

Pseudo-Animation

已处理的位置　　未处理的位置



Pseudo-Animation

已处理的位置　　未处理的位置

Actual-Animation

已处理的位置

未处理的位置

递归处理

例题（陈立杰）

- 给定平面上n个点的坐标，每个点有两个权值R[i]和C[i]
- 点i到点j有边当且仅当i<j且点i到点j的欧几里得距离 < R[j]
- (即i<j且在点i在点j的圆内)
- 求一条路径，使C[i]和最大

预备知识

- 给定大小为n的点集A，我们能够在O(n log n)的时间复杂度内建立一个数据结构，支持在O(log n)的时间复杂度内查询给定的一个点B到点集A中点的最短距离

- 例：Voronoi图+点定位数据结构

思路

- 简单的平方算法：直接DP
- F[i]为以i为终点的路径的最大C值和

- 在线数据结构维护：不会做（?）

分治算法

- 转化原问题
- 已处理点集A，待处理点集B
- 对于B中每个点j，求A中在圆j内的点最大F值

再次转化

- 将A中点按照F值排序
- 二分F值最大的点，判断F值不大于它的点中到j的最短距离是否小于R[j]

线段树

- 将A中点按F值排序后建立线段树
- 每个区间内存储该区间内所有点的Voronoi图以及点定位数据结构

- 二分时每次取的是线段树的区间端点，因此对于B中每个点，事实上只查询了线段树的O(log |A|)个区间。

复杂度分析：时间复杂度

- Voronoi图+点定位数据结构：O(k log n)
  其中k为Voronoi图点数
- 线段树：O(|A| log^2 n)
- Sum |A| = n log n
- 建立总复杂度 O(n log^3 n)

复杂度分析：时间复杂度

- 点定位查询 O(log n)
- 线段树查询结点个数：O(log n)
- 每个结点查询的次数：O(log n)
- 查询总复杂度 O(n log^3 n)

- 总复杂度：O(n log^3 n)

复杂度分析：空间复杂度

- 每次只需要存储一棵线段树
- 最大的线段树 $|A|=O(n)$
- 每个区间存储的数据结构的空间为O(区间长度)
- 因此总空间复杂度为$O(n \log n)$

必须离线？

- 注意到我们这里的转移方式并不要求将B中的所有点一起转移
- 可以改变转移顺序使得转移成为在线（即，可以在末尾增加一个点）
- 使用空间会增加，但是空间复杂度不变
- 详见讲义

不仅是动态规划

- “动态”问题
- n个状态(操作)，每个状态(操作)对后面的状态产生影响

- 只能动态做?

动态最小生成树

- 维护一个图，每次修改一条边的权值(可以增加或减少)，并返回最小生成树的边权和
- 所有修改预先给出，即可以接受离线算法
- (本题支持加边删边，只要允许正无穷边存在即可)

思路

- 暴力：单次修改O(n)

- 在线算法：非常复杂，目前已知的最优算法，单次修改O(log^4 n)

分治算法

- 分治算法的根本是减少数据量
- 需要修改的边数是可以接受的
- 瓶颈在于图的点数和边数

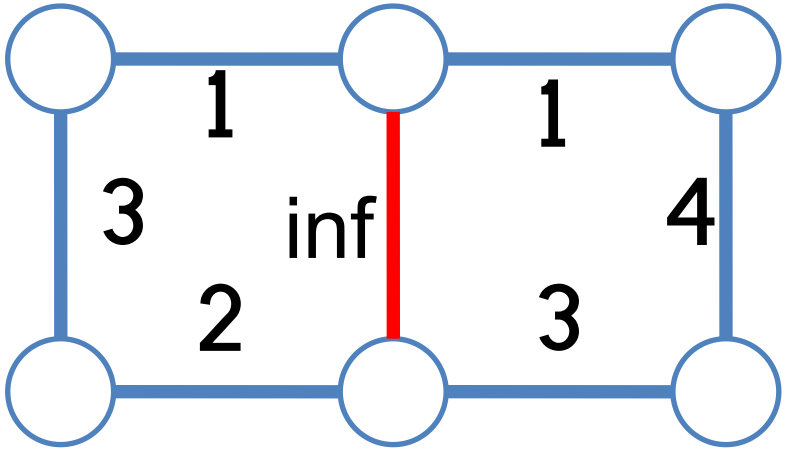
思路

- 有一个带边权图G，有k条边在之后会被修改 （并询问MST）
- 在这k个MST中，有一些边是永远不会在MST中的；而另一些边是必须在MST中的
- 我们可以试图求出这些边，以缩小图的规模

Reduction（无用边）

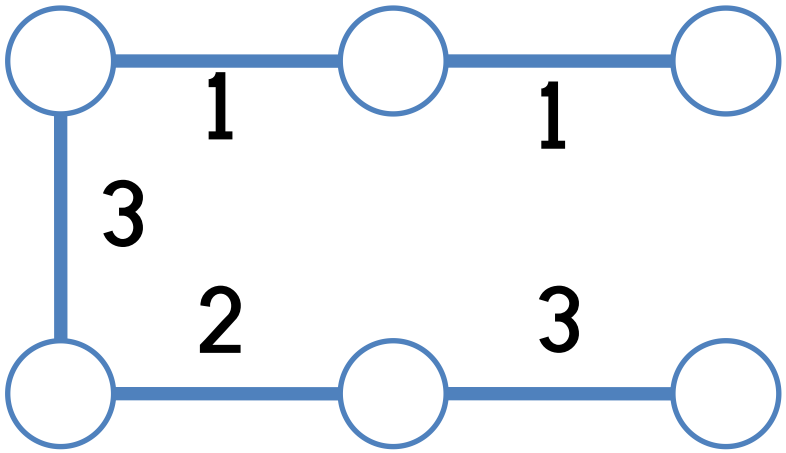
- 将需要修改的边边权标记为正无穷，做MST
- 原图中边权非正无穷且不在MST内的边，在还原边权后必然也不在MST内
- 删除这些边，减少边数
- 还原边权

Pseudo-Animation



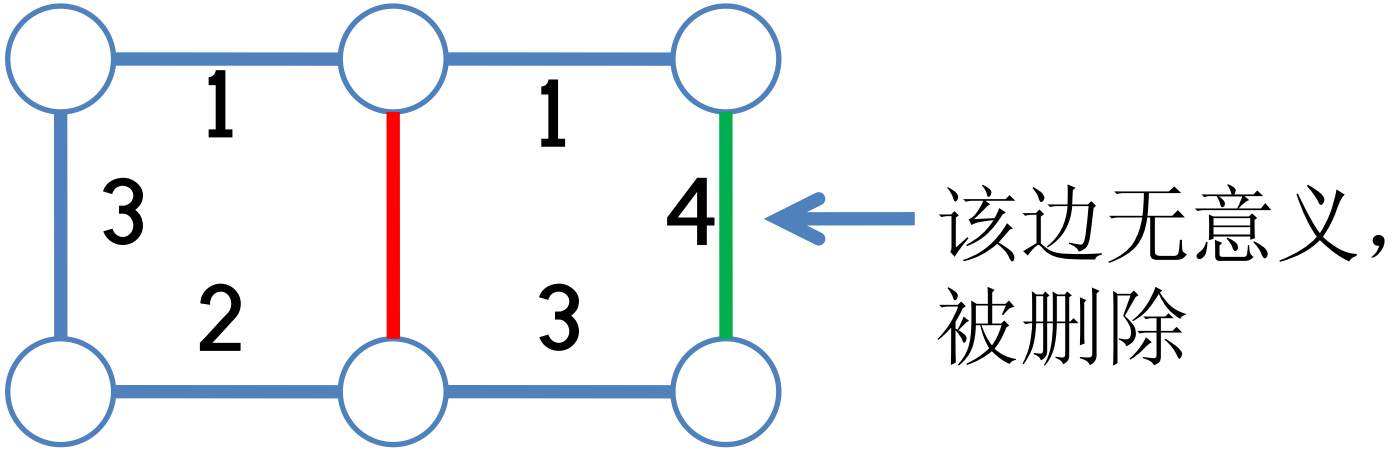
红边为待修改边

Pseudo-Animation

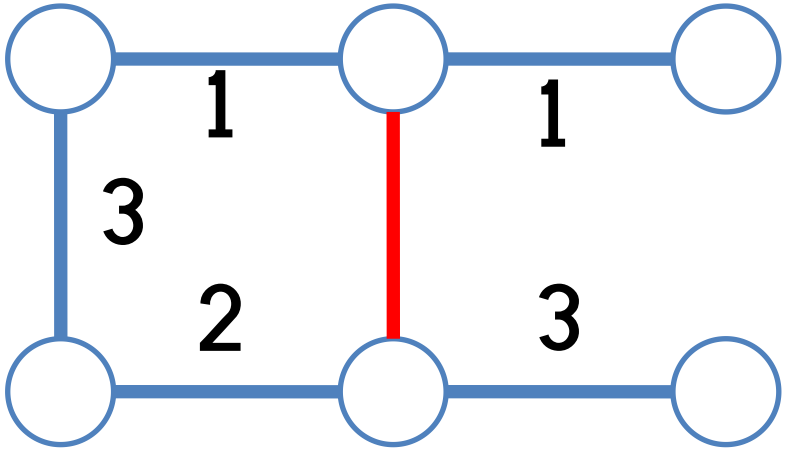


最小生成树

Pseudo-Animation



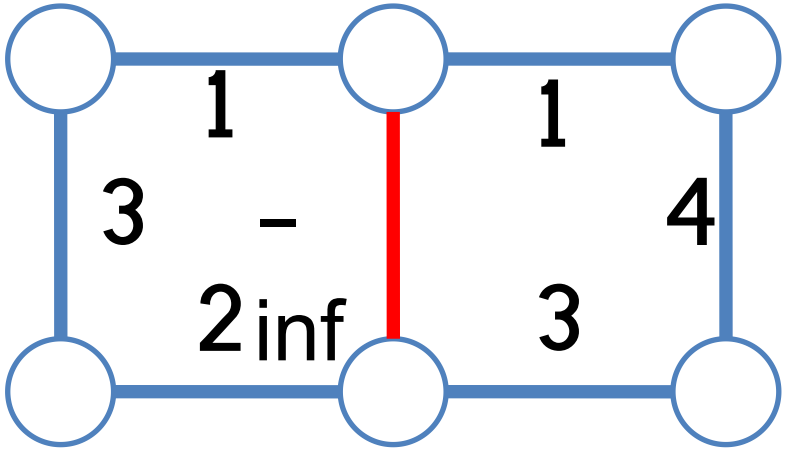
Pseudo-Animation



Contraction（必须边）

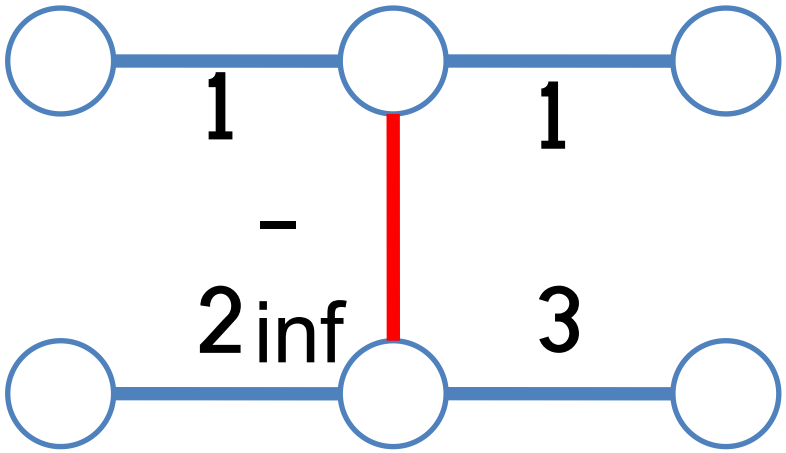
- 将需要修改的边边权标记为负无穷，做MST
- MST中非负无穷边，还原边权后必然也在MST内
- 将这些边连接的点集合并，缩小点数
- 还原边权

Pseudo-Animation



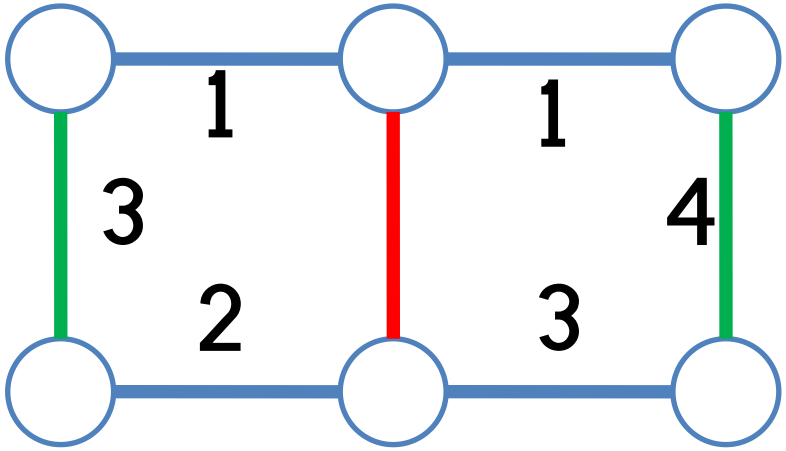
红边为待修改边

Pseudo-Animation



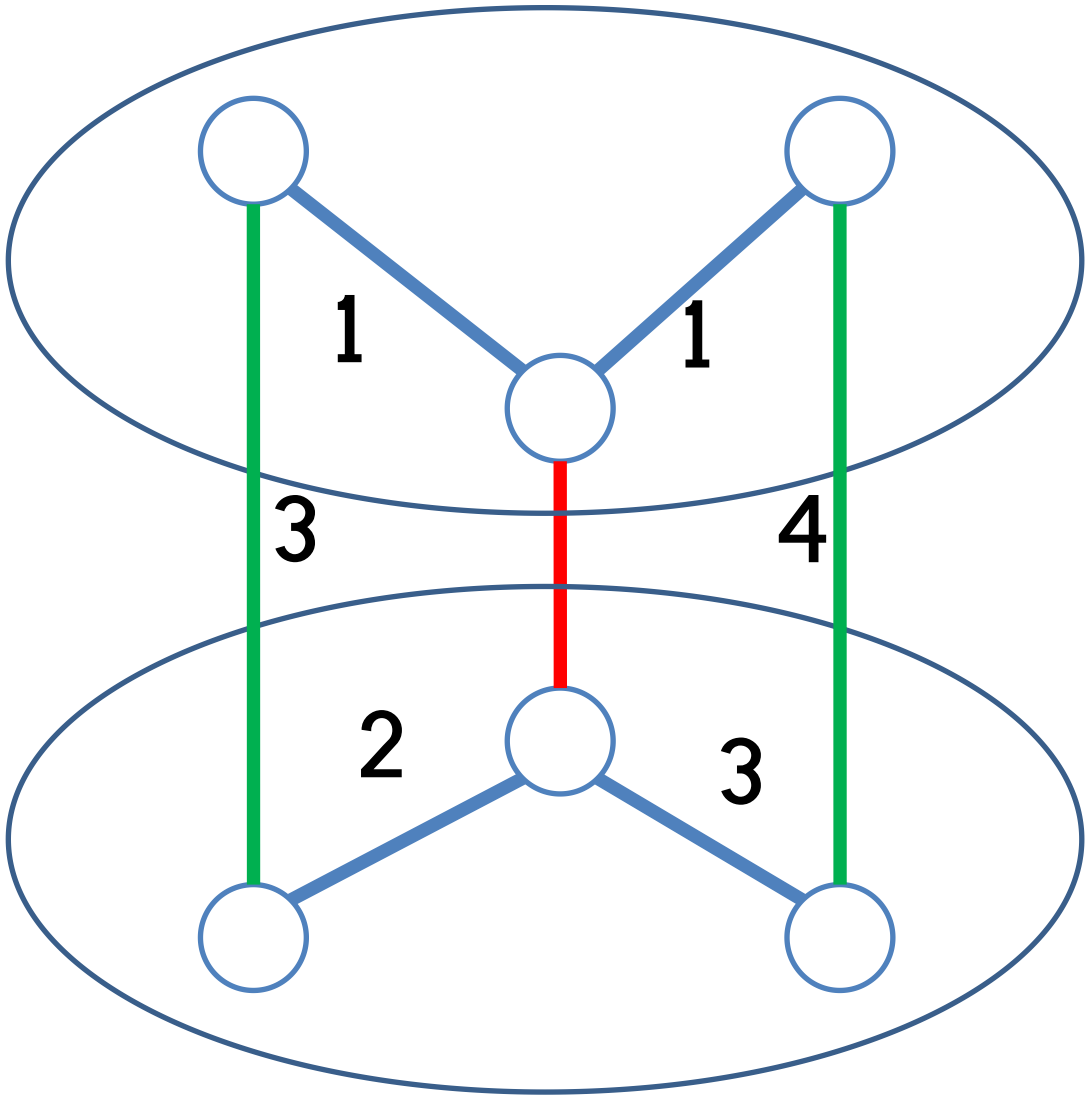
最小生成树

Pseudo-Animation



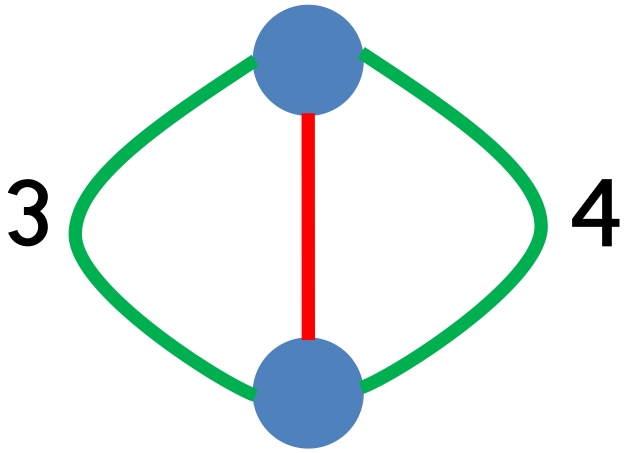
四条蓝边为必须边

Pseudo-Animation



缩点

Pseudo-Animation



缩点

减少点数和边数

- Reduction−Contraction−Reduction

- 假设当前区间内修改的边数为k
- 进行R−C−R后，图中最多只剩下k+1个点和2k条边
- 可以接受

简单的证明

- 初始：n个点m条边 k条待修改边
- Reduction后：最多剩余n+k-1条边
- Contraction后：最多剩余k+1个点

- R-C-R过程：
- (n, m) => (n, n+k-1) => (k+1, n+k-1) => (k+1, 2k)

流程

- 当前处理区间[L，R)，中点M
- 若L=R−1，则此时点数和边数均不超过2，直接实行修改，记录答案并退出
- 删边、缩点
- 递归处理[L，M)
- 递归处理[M，R)

时间复杂度

- 假设当前需要处理的区间长度为k
- simplify需要 O(k log k + k α(k))
- 实际上，O(k log k)的排序部分可以通过归并排序变成O(k)
- 总时间复杂度?
- 根据主定理只能知道 T(n)=Ω(n log n) 以及 T(n)=o(n log^(1+eps) n)

其他题目

- BOI 2007 Mokia
- POI 2011 Meteors
- 2012年集训队互测 梁盾 矩阵乘法

总结

- 动态问题：n个状态(操作)，每个状态(操作)对之后的状态施加影响
- 分治算法：通过分治，处理看似难以处理的影响
- 1D1D动态规划的分治算法：将连续一段影响同时处理
- 动态MST的分治算法：减小数据量，减少实施影响的花费

优势与劣势

- 可以采用离线数据结构
- -> 存在性、代码量、复杂度

- 题目必须允许离线（?）

Thanks

例题一：在线维护？

- 我确实不会做 = =
- 不过这里有一些思路

(伪)动态Voronoi图

- 支持：加点、询问（均在线）
- 还记得例题一的做法实际上是在线的么？

二项堆、线段树与分治

- 只添加结点的二项堆
- 从左向右每个点分别查询并插入、实时删除无用结点的线段树
- 分治：A集合中点更新B集合中点，A集合中点建立数据结构后B集合中点的值可以在线查询

- 三者是等价的